Visual Studio Magazine Online
Classic VB Corner

Persisting Environment Variables

ONLINE ONLY

Creating and persisting new environment variables can be surprisingly tricky in Classic VB. Here's how to avoid a common trap.

August 11, 2009 · by Karl E. Peterson

A reader who was having fun writing console apps, asked me recently how he might be able to modify environment variables (e-vars) so that subsequent instances of his apps would see the changes. He pointed out, correctly, that this is pretty easy with batch (or CMD) files, and wanted to be able to do the same thing with his ClassicVB console apps.

Well, the long and the short of it is, you can't. At least, not exactly. But you can come pretty close. It's not all that hard to change e-vars and have them persisted into the future. To do so, you create new entries under either HKCU\Environment for user-scoped variables, or HKLM\System\CurrentControlSet\Control\Session Manager\Environment for machine-scoped (affects all users) variables.

There are any number of Web sites out there that will then tell you that to propagate these newly set e-vars to the parent process, you simply need to send a WM_SETTINGCHANGE of "Environment" to HWND_BROADCAST. Sounds pretty simple, and it certainly must be true given how many times it shows up in a quick Google search, right?

Not So Fast

It turns out that following this procedure will indeed create and persist new environment variables. And if the parent process actually gives a rip about WM_SETTINGCHANGE broadcasts, it may choose to update how it looks at the environment. The problem is, the most common command-processor out there -- cmd.exe -- doesn't pay any attention to this message. You can create new e-vars in one command window, then see them if you open a new command window. But they simply do not show up in the original.

That needed to be said, because the implication is misleading. Even Microsoft's advice lacks clarity on this point.

And yet, it's still useful to know how to set, clear or modify e-vars, so let's take a closer look at that. Here's the test app I wrote, to see how this might work:

```
Public Sub Main()
   Dim nResult As Long
   Const lmKey = _
     "System\CurrentControlSet\Control\Session Manager\Environment"
   Const cuKey = "Environment"
   Const lmEvar = "ClassicVB-lm"
   Const cuEvar = "ClassicVB-cu"

   ' Required in all MConsole.bas supported apps!
   Con.Initialize

   ' Check whether to clear or set variables.
   Con.WriteLine "Writing to registry...  ", False
   If InStr(1, Command$, "/clear", vbTextCompare) Then
      ' Clear e-vars at the User and Machine levels.
      ' HKLM calls may fail if not running as admin.
      Call RegDeleteValue(HKEY_CURRENT_USER, cuKey, cuEvar)
      Call RegDeleteValue(HKEY_LOCAL_MACHINE, lmKey, lmEvar)
   Else
      ' Set e-vars at the User and Machine levels.
      ' HKLM calls may fail if not running as admin.
      RegSetStringValue HKEY_CURRENT_USER, cuKey, cuEvar, "Rocks!"
      RegSetStringValue HKEY_LOCAL_MACHINE, lmKey, lmEvar, "Rocks!"
   End If

   ' Tell the world what we've done.
   Con.WriteLine "Broadcasting change..."
   Call SendMessageTimeout(HWND_BROADCAST, WM_SETTINGCHANGE, 0&, _
      "Environment", SMTO_ABORTIFHUNG, 5000, nResult)

   ' Demonstrate success, or lack thereof
   Con.WriteLine cuEvar & "=" & Environ$(cuEvar)
   Con.WriteLine lmEvar & "=" & Environ$(lmEvar)

   ' Allow user to see output if launched from Explorer.
   If Con.LaunchMode = conLaunchExplorer Then
      Con.PressAnyKey
   End If
End Sub
```

This uses what I wrote about in my last column, vbAdvance and my MConsole module, to create a true console application. I've added the demo to the Console sample on my site.

If you run that app at the command line, within a single command window, it will never show (using the Environ$ function) the changes you're making. Nor do subsequent processes started in the same command window, as they inherit copies of the original environment block rather than the altered one that cmd.exe didn't bother to read when notified. But if you were to double-click the EXE successively in Explorer, the second instance would see the changes made by the first.

This all leads one to conclude that setting e-vars in a console app is really rather pointless, unless the application is written with the intention of making long-term settings changes. For example, as part of a setup procedure this can make good sense. But setup programs are rarely console based, so we may as well have a more

generic set of routines to make lasting e-var changes. We can write a more generic routine that looks something like this:

```
Public Function eVarWrite(ByVal eVar As String, _
   ByVal eVal As String, Optional ByVal HKLM As Boolean = False, _
   Optional ByVal Expandable As Boolean = True) As Boolean

   Dim RootKey As Long
   Dim SubKey As String
   Dim dwType As Long
   Dim nRet As Long
   Dim hKey As Long

   ' Is this user-specific or machine-wide?
   If HKLM Then
      RootKey = HKEY_LOCAL_MACHINE
      SubKey = hklmSubKey
   Else
      RootKey = HKEY_CURRENT_USER
      SubKey = hkcuSubKey
   End If

   ' Allow for variable expansion, by default.
   If Expandable Then
      dwType = REG_EXPAND_SZ
   Else
      dwType = REG_SZ
   End If

   ' Open a key and set a value within it.
   If apiRegOpenKeyEx(RootKey, SubKey, 0&, KEY_ALL_ACCESS, hKey) _
      = ERROR_SUCCESS Then

      ' Attempt to write data - Always a string.
      nRet = apiRegSetValueEx(hKey, eVar, 0&, dwType, _
             ByVal eVal, Len(eVal))
      Call apiRegFlushKey(hKey)
      Call apiRegCloseKey(hKey)
      ' Return result of RegSetValueEx call.
      eVarWrite = (nRet = ERROR_SUCCESS)
   End If
End Function

Public Function eVarAlert() As Long
   ' This can take a few seconds, so it makes sense to have
   ' it in a separate routine and only call it after making
   ' all environment variable changes.
   Call SendMessageTimeout(HWND_BROADCAST, WM_SETTINGCHANGE, 0&, _
      "Environment", SMTO_ABORTIFHUNG, 5000, eVarAlert)
End Function
```

Variable Expansion

I decided it best to separate the setting of e-vars with the broadcast of that change, as it routinely takes several seconds for the notification message to be sent. You can view the complete module, with full declarations, on my Web site.

If you've never worked with variable expansion before, it's kind of cool. You can, for example, set an e-var to a value like "%temp%", and then whenever a program queries that value it's automatically expanded to whatever value the "temp" e-var currently holds. That may sound confusing, but here's what it would look like at the command prompt:

```
C:\›set test=%temp%

C:\›set test
test=C:\WINDOWS\TEMP
```

And yet, if you were to go in and look at the HKCU\Environment key, you'd see the "test" e-var does indeed hold a value of "%temp%". Very useful, for situations where variables contain sub-values that are themselves variable.

Getting back to the original question, I'm afraid that the best way to truly affect the parent process, if it's going to be cmd.exe, would be for the ClassicVB app to create a CMD file that's executed immediately following itself as part of the batch process. In that case, it's really the parent process that's altering itself, so that will always work.

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.